Fábrica de Noobs

Desmistificando: problema de 2038

Natanael Antonioli

Janeiro de 2026

1. SUMÁRIO

2. Com o que estamos lidando? 3

3. Um recado especial... 4

4. O que são a data *Unix* e o padrão de 32 bits? 6

5. Qual o problema ao combinar data *Unix* com 32 bits? 8

6. Como *hardware* e linguagem de montagem serão afetados? 9

7. Como resolver o problema a nível de *firmware* e sistema operacional? 11

8. Como resolver o problema a nível de *software*? 13

9. Conclusões 16

2. COM O QUE ESTAMOS LIDANDO?

Em 2038, é esperado que um problema de data quebre todas as aplicações que usem a data *unix* em 32 bits, o que corresponde a grande parte da tecnologia atual e é projetado por alguns de causar transtornos globais.

A reprodução da história fica por conta do portal R7 (https://noticias.r7.com/tecnologia-e-ciencia/bug-pode-fazer-computadores-pararem-em-2038-18122014/).

Figura 1: reprodução da história.

Noticias R7 > Tecnologia e Ciência

Bug pode fazer computadores pararem em 2038

Motivo seria limitação na contagem de valores dos processadores 32-bit

TECNOLOGIA E CIÊNCIA | Do R7
18/12/2014 - 12H36 (ATUALIZADO EM 15/08/2025 - 11H56)

3. UM RECADO ESPECIAL...

Esse vídeo trata de um problema real que afeta múltiplas linguagens de programação e sistemas da informação, um tema que muitos tem interesse, que é fundamental para o mercado de trabalho e que pode parecer intimidador.

Pensando nisso, o canal *Fábrica de Noobs*, em parceria com a *Hashtag Treinamentos*, oferecerá, através de um link exclusivo e com 500 vagas reservadas para o nosso público, **um treinamento de 4 dias em linguagem *Python* para iniciantes**, que será **100% gratuito** e não necessita de **nenhum pré-requisito**.

No treinamento, que se diferencia de cursos usuais por possuir **apostila**, **suporte a dúvidas** e **certificado de participação**, trará projetos práticos em automatização de tarefas, análise de dados e criação de *chatbots* com inteligência artificial, que podem servir como empurrão inicial em uma carreira de desenvolvedor e integrar um portfólio.

Inscritos do canal **podem participar pelo link exclusivo** (https://lp.hashtagtreinamentos.com/python/jornada/inscricao?fonte=fabricadenoobs&utm_campaign=fabricadenoobs&utm_medium=fabricadenoobs&utm_source=Influencer&utm_content=jornadaphyton26&curso=python) que se encontra também na descrição e no comentário fixado, ou do *QR Code* que se encontra abaixo.

Figura 2: *QR Code* para acesso ao treinamento.

A *Hashtag Treinamentos* possui mais de 120 mil alunos formados, com índice de satisfação acima de 98%, e múltiplos casos de sucesso documentados em que alunos, com a ajuda dos treinamentos fornecidos, **saíram do zero ao mercado de trabalho em pouco tempo**.

Através de um conhecimento sólido em linguagens de programação – sendo *Python* uma das mais indicadas para iniciantes – pode-se compreender como sistemas de informação funcionam não só em aspectos como o problema de 2038, mas em aplicações reais utilizadas na indústria.

4. O QUE SÃO A DATA *UNIX* E O PADRÃO DE 32 BITS?

Um dos principais aspectos da computação é o armazenamento de dados. Alguns desses dados, como data e hora, **representam desafios adicionais** por diversas razões:

- São compostos de **ao menos seis valores**: segundo, minuto, hora, dia, mês e ano;
- Operações aritméticas **não são triviais**: a soma ou subtração envolve operar em várias unidades, sendo que cada uma é um múltiplo diferente da inferior;
- Assumem **múltiplas representações**: um mesmo dia e hora tem valores diferentes a depender do fuso-horário.

Uma solução para armazenar data e hora que alivia o programador desses problemas **é a data *Unix***, que armazena apenas um valor inteiro em segundos para **representar o total de segundos depois das 00:00:00 do dia 1 de janeiro de 1970 no fuso-horário UTC**, em que cada dia transcorrido representa 86400 segundos.

Computadores, porém, **armazenam valores em números binários**. Números binários são sequências de zeros e uns, que representam coeficientes de potências de 2. Por exemplo, podemos fazer:

$$101011_2 = 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 1 \times 2^5$$

$$= 1 + 2 + 0 + 8 + 32 = 43_{10}$$

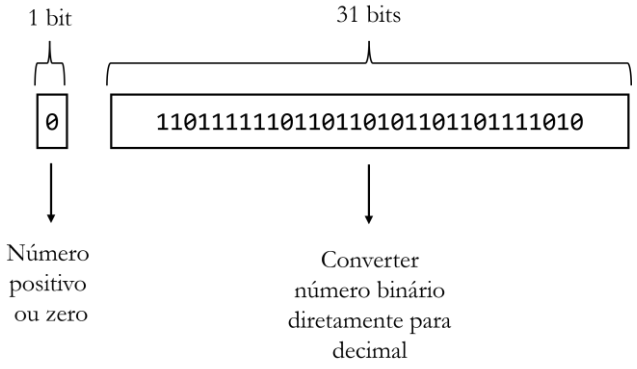
Em um computador, um número binário **é armazenado em unidades de memória** que possuem um tamanho geralmente fixo dentro de uma arquitetura – ou seja, que restringe o tamanho máximo de um número binário que pode ser processado por ciclo.

Um modelo muito popular nos anos 2000 **foi o de 32 bits**, em que cada número binário é armazenado em um espaço com **32 unidades que podem assumir valores zero ou um**. Assim, o total de combinações possíveis de serem armazenadas é de 2^{32}.

Na prática, a forma de armazenamento dos dados é em complemento de 2, em que **o bit mais significativo existe para definir o sinal do número armazenado nos outros 31 bits**.

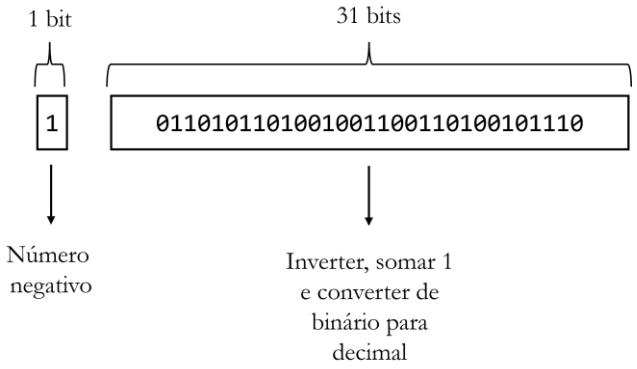
Caso o número seja zero ou positivo, o bit de sinal assume o valor 0, e os próximos 31 bits devem simplesmente ser convertidos para decimal usando o algoritmo que conhecemos.

Figura 3: número positivo em complemento de 2.



Caso o número seja negativo, porém, o bit de sinal assume o valor 1, e a conversão para decimal requer que, primeiro, o número de 31 bits seja invertido (trocando 0 por 1, e vice-versa) e, depois, somado ao valor 1. Aqui, indicamos datas antes da data padrão.

Figura 4: número negativo em complemento de 2.



5. QUAL O PROBLEMA AO COMBINAR DATA *UNIX* COM 32 BITS?

Na data *Unix*, a soma ou subtração de qualquer período **é sempre feita somando ou subtraindo o equivalente em segundos**. Por exemplo, para subtrair 2 dias e 1 hora de um registrador, devemos apenas subtrair $86400 \times 2 + 3600 = 176400$ desse registrador. Todas as nuances (como mudança de mês e horário de verão) são tratadas por outros algoritmos.

Logicamente, a cada segundo, um processador de 32 bits operando com data *Unix* **irá somar 1 ao valor armazenado previamente em 32 bits**. Isso não é um problema até escrevermos, nesse registrador, o maior número possível de 32 bits em complemento de 2, que é:

$$01111111\ 11111111\ 11111111\ 11111111_2 = 2.147.483.647_{10}$$

Esse valor corresponde a 2.147.483.647 segundos depois das 00:00:00 do dia 1 de janeiro de 1970, que **é equivalente às 03:14:07 do dia 19 de janeiro de 2038**. O que acontece quando, no segundo seguinte, o processador somar 1?

O número sucessor ao binário $01111111\ 11111111\ 11111111\ 11111111$ é, naturalmente, o binário $10000000\ 00000000\ 00000000\ 00000000$.

$$01111111\ 11111111\ 11111111\ 11111111_2 + 1_2 =$$

$$= 10000000\ 00000000\ 00000000\ 00000000$$

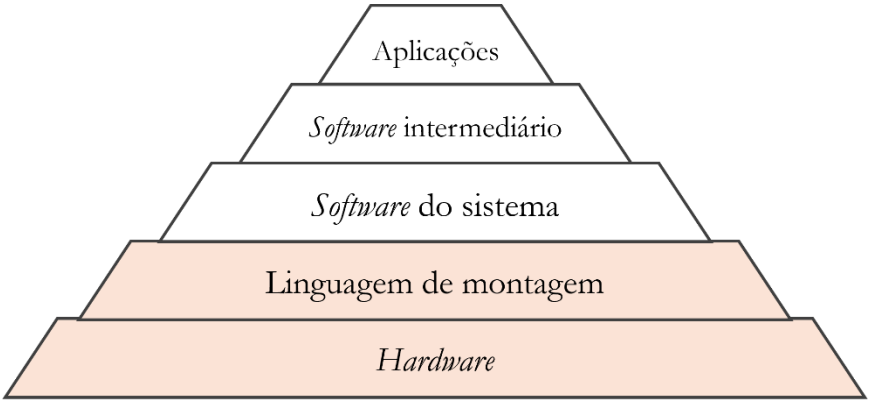
Porém, esse binário passa a possuir o dígito mais significativo 1, o que significa que ele se tornará negativo. E essa leitura nos indica $-2.147.483.648$, ou seja, **aproximadamente 2 bilhões de segundos antes da data base**, o que é equivalente **às 20:45:52 do dia 13 de dezembro de 1901**.

Logo, o problema do ano 2038 surge quando um processador de 32 bits usando a data *Unix* **salta da data correta para dezembro de 1901 ao tentar avançar 1 segundo**.

6. COMO *HARDWARE* E LINGUAGEM DE MONTAGEM SERÃO AFETADOS?

Na computação, é comum dividir elementos em diversos níveis de abstração. É seção trabalha com os impactos do problema de 2038 em baixo nível, que compreende *hardware* (ou seja, componentes físicos) e instruções de máquina, que são sequências binárias interpretadas pelo *hardware* como comandos e lidas por seres humanos como linguagem de montagem.

Figura 5: níveis discutidos nessa seção.



Componentes físicos **não sabem o dia e hora atual** e, quando armazenamos essa informação em formato de data *Unix*, também **não são capazes de compreender o que estão armazenando**.

Para um componente físico, uma data *Unix* **é simplesmente um valor armazenado na notação de complemento de dois**, que o componente é capaz de operar, por exemplo, na forma de adição.

O componente operará corretamente em 2038, e, se possuir 32 bits, quando somado em 1 produzirá o seguinte (que sabemos ser $10000000\ 00000000\ 00000000\ 00000000_2$) e, a depender de sua arquitetura, **produzirá um sinal indicando que a soma extrapolou o limite**, chamado de *overflow*.

A nível de comandos (que aqui serão representados em linguagem de máquina, ou *assembly*), esse tipo de operação, em um registrador de 32 bits, poderá trazer efeitos indesejados.

Por exemplo, o código abaixo soma `1` a uma variável chamada `time_today` e, então, compara com uma data limite, que, se maior, avança para um setor do código chamado `tomorrow` e, caso contrário, para um setor chamado `yesterday`. Em 2038 e com uma combinação crítica de valores, **esse código poderá tomar a decisão errada porque o registrador voltou para 1901**.

```
mov eax, [time_today]     ; carrega o tempo atual
add eax, 1                ; soma 1 ao tempo atual

mov [time_today], eax     ; carrega novamente

cmp eax, [limit_date]     ; compara com a data limite
ja  tomorrow              ; se sim, avança

jmp yesterday             ; se não, retroage
```

Em nível de hardware, a melhor prevenção do problema de 2038 é **a substituição dos componentes** (como processadores e memórias) de 32 bits **por componentes equivalentes de 64 bits**.

Isso não resolve o problema de maneira definitiva, afinal, registradores de 64 bits em complemento de 2 também têm um valor máximo que pode ser armazenado. Felizmente, esse valor **equivale a cerca de 292 bilhões de anos**, o que é 21 vezes a idade atual do universo, e tempo suficiente para não mais nos preocuparmos com o problema – o colapso do Sol será uma preocupação muito antes disso.

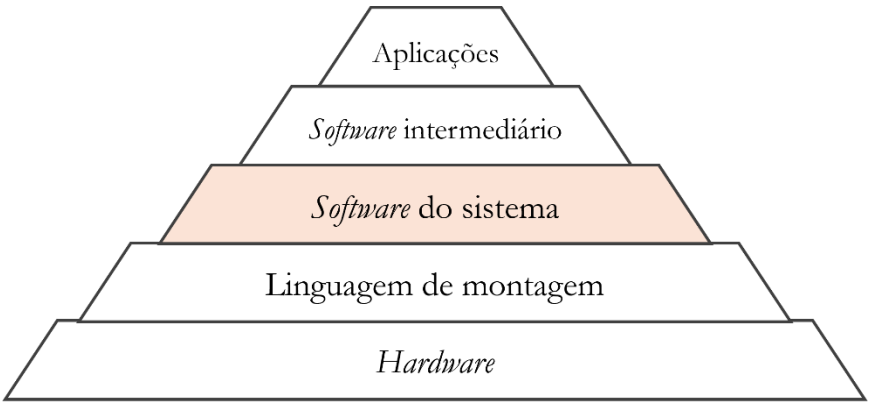
Novos dispositivos computacionais para o consumidor final (como processadores de *desktop* e *smartphones*) **já costumam utilizar 64 bits**, o que inclui a série de processadores *Intel Core i3*, *i5*, *i7* e *i9*; todos os processadores da série *AMD Ryzen*; todos os *iPhone* a partir do *s5* (de 2013) e a maioria dos celulares com *Android* superior ao *5*

No entanto, alguns dispositivos fabricados sem considerar o problema do ano 2038 **podem ainda estar em operação em 2038**, e não é prudente simplesmente assumir que esses dispositivos já terão sido descartados quando esse ano chegar. Se uma solução em nível mais alto não for possível, **é necessário substituir esses componentes**.

7. COMO RESOLVER O PROBLEMA A NÍVEL DE *FIRMWARE* E SISTEMA OPERACIONAL?

Chamamos de *firmware* um software de extremo baixo nível, responsável por realizar a ponte entre o *hardware* e o sistema operacional, que é de propósito geral e oferece aos *softwares* um conjunto de comandos que os permitem aproveitar os recursos do computador.

Figura 6: níveis discutidos nessa seção.



Felizmente, a maioria dos *firmwares* **não utilizam a data *Unix* para contagem de tempo**, mas sim outros contadores internos e, portanto, não são afetados pelo problema. Ainda assim, caso isso ocorra, **eventos escalonados podem ser executados fora de ordem** e cheques de validade, como certificados de segurança, **podem falhar**, o que causa transtornos maiores.

Um *firmware*, porém, **pode ser atualizado para que a data seja armazenada em 64 bits**, por exemplo, **utilizando dois registradores de 32 bits**. Porém, mudanças do *firmware* surtirão pouco efeito para solucionar o problema, já que a maioria das mudanças deverão ser feitas no sistema operacional. No sistema operacional, o núcleo define uma variável de data e hora em data *Unix* da seguinte forma.

```
time_t signed 32-bit integer
```

Isso é um problema porque tal valor é passado para múltiplas funções (como `time()` e `stat()`), sistemas de arquivos, temporizadores, escalonamento e protocolos de rede. A mudança abaixo resolve o problema, mas se for feita sozinha, **quebra todo o resto do sistema operacional** porque todo o código foi produzido a partir da premissa de que o tempo é armazenado em 32 bits.

```
time_t signed 64-bit integer
```

A solução é, além dessa mudança, refazer **todas as chamadas de sistema, todas as estruturas de dados que envolvam tempo e todo o sistema de arquivos** para comportar tempo em 64 bits.

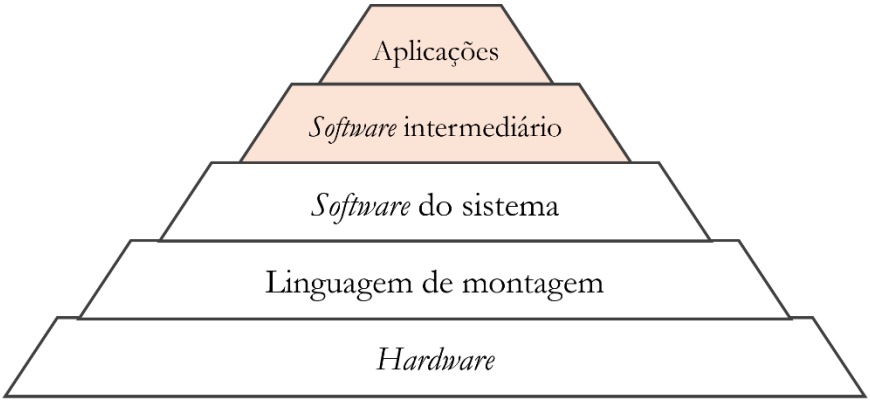
No entanto, uma mera mudança do tipo **causa incompatibilidade com *softwares* antigos**, o que significa que o sistema operacional deverá ser capaz de não só operar em 64 bits, mas **receber valores em 32 bits, lidar com possíveis erros, e devolvê-los de forma consistente**.

8. COMO RESOLVER O PROBLEMA A NÍVEL DE *SOFTWARE*?

Chamamos de *middleware*, ou *software* intermediário, aqueles *softwares* que existem para tornar a escrita de aplicações alto nível mais fácil para o programador, que apenas escreve comandos em uma linguagem, e permite que o *middleware* use os recursos oferecidos pelo sistema operacional executá-lo. Assim, é fácil produzir aplicações finais.

Em *middleware*, classificamos múltiplas linguagens de programação, tanto de baixo nível (como *C*, em que o programador precisa se preocupar, por exemplo, com alocação dinâmica de memória) como de alto nível (como *Python*, que não expõe tais problemas ao programador).

Figura 7: níveis discutidos nessa seção.



O *middlware* costuma herdar certas definições do sistema operacional, o que significa que o comando abaixo poderá definir `long` como 32 ou 64 bits, a depender da arquitetura no nível inferior.

```
typedef long time_t;
```

Assim, o *middleware* precisa ser refeito para herdar um valor de 64 bits do sistema operacional e, então, **trabalhar internamente com ele**.

Ainda assim, como em todos os outros casos, o *middleware* **não pode abandonar o suporte a 32 bits repentinamente**, e é possível que alguns jamais poderão fazer isso, precisando manter duas APIs: uma destinada a programadores trabalhando com 64 bits, e uma destinada a programas antigos que operam em 32 bits.

Isso ocorre porque muitos continuarão a utilizar aplicações que não mais são mantidas, o que significa que, ainda que todo programador passe a programar com as novas APIs de 64 bits, aplicações antigas **continuarão a necessitar de uma API de 32 bits**.

Um exemplo está no código abaixo, em linguagem *C*, em que, no padrão antigo, três números de 32 bits são armazenados.

```
#include <stdint.h>
typedef struct {
	int32_t created_at;
	int32_t updated_at;
	int32_t expires_at;
} AssetMeta;
```

É razoável concluir que, no padrão novo, isso se resuma a trocar o tipo de dado. Porém, agora, uma função antiga que procure por `updated_at` irá acessar `created_at`, já que ela procurará pela posição `33`, mas as posições de `0` a `63` na estrutura nova pertencem à `created_at`, e `updated_at` só começa na posição `64`.

```
typedef struct {
	int64_t created_at;
	int64_t updated_at;
	int64_t expires_at;
} AssetMeta;
```

No presente, múltiplos *middlewares* **já foram corrigidos para o problema de 2038**, como *Ruby* (desde 2019), *NetBSD* (desde 2012), *OpenBSD* (desde 2014), *C* (desde 2021), *MySQL* (desde 2022), *MariaDB* (desde 2024) e *Visual C++* (desde 2005). Outros, *Java* e *Javascript*, **nunca foram vulneráveis**, ao passo que *Python*, por depender de C, **é seguro em sistemas de 64 bits**.

Ainda assim, um programador de aplicações nessas linguagens **não deve considerar seu trabalho previamente feito**: é possível que aplicações ainda utilizem estruturas de dados que dependam de comandos em 32 bits da linguagem em que foram desenvolvidas, **o que necessita correção**, especialmente porque muitas linguagens **farão essa mudança de forma explícita**.

Além disso, muitas aplicações **usam arquivos e bancos de dados produzidos com valores de data em 32 bits**, e precisam ser refeitas para lidar com esses arquivos, **convertendo-os para o novo formato** e evitando que informações no formato antigo causem comportamentos inesperados.

9. CONCLUSÕES

Assim, podemos concluir que **o problema de 2038 é real** e afetará quaisquer componentes computacionais que armazenem data no formato *unix*, mas **é improvável de causar transtornos globais** porque **já está sendo solucionado**.

No entanto, **não existe uma solução universal**, e nenhum trabalho feito em níveis inferiores de abstração **é suficiente para dispensar programadores em níveis superiores** de garantir que suas aplicações são seguras.

Além disso, as partes interessadas no correto funcionamento de *softwares*, como empresas e governos, **deverão garantir que estão utilizando ferramentas seguras para o problema de 2038**.